Your first libpari function

A tutorial

A. Page

IMB
CNRS/Université de Bordeaux

23/06/2025

Goal

The goal of this tutorial is to implement a very simple function:

```
interval(a,b): the integer interval [a..b].
```

You will then have a basic function to start playing with the concepts explained in the other tutorials. Do not worry if you do not understand everything, it will be clarified in the next tutorials!

Branch

In order to develop Libpari code, you need to use git. Let us create a branch for this tutorial.

```
git checkout -b tuto-first
```

Initialise tests

I personally prefer writing tests before the corresponding code.
This way, one produces testable code and the code is always tested.
Create the file

```
src/test/in/interval
```

containing a single line:

```
interval(1,0) == []
```

Then update configuration so the test file is taken into account:

```
./Configure -l
```

Run test

Now run the test:

```
make test-interval
```

Of course, the test fails. Read the output in the file:

```
Olinux-x86_64/interval-sta.dif
```

Installation/documentation

Determine the specification of functions before implementing them!
Create the file

`src/functions/linear_algebra/interval`

containing the following:

```
Function: interval
Section: linear_algebra
C-Name: interval
Prototype: GG
Help: interval(a,b): return the integer interval [a..b].
Doc: Return the integer interval~$\{a,a+1,\dots,b\}$.
 Empty if~$b<a$.
```

Try compiling:

```
make gp
```

Starting the C code

Edit the header file

`src/headers/paridecl.h`

by adding the lines

```
/* interval.c */
GEN     interval(GEN a, GEN b);
```

Create a file

`src/basemath/interval.c`

...

Starting the C code

... containing the following:

```
#include "pari.h"
#include "paripriv.h"

GEN interval(GEN a, GEN b)
{
  pari_sp av = avma;
  return gc_GEN(av, gen_0);
}
```

The garbage collection is unnecessary here, but I prefer writing it immediately so as not to forget after adding more code.

Update the configuration for the new file and compile:

```
./Configure -l
make gp
```

More testing

We now have code that compiles! Let's test it!

```
./gp
? interval(1,0)
% = 0
```

Not correct, the code is not doing anything yet!

```
make test-interval
```

But the test passes, because `0==[]`.

Add a new test:

```
type(interval(1,0))=="t_VEC"
```

This test fails.

Developing the code

Edit the C code to actually create a vector:

```
GEN interval(GEN a, GEN b)
{
  pari_sp av = avma;
  GEN v;
  v = cgetg(1, t_VEC);
  return gc_GEN(av, v);
}
```

This makes our test pass. Let's add a more interesting test:

```
interval(0,1) == [0,1]
```

Developing the code

Edit the C code to create a vector of the correct length:

```
GEN interval(GEN a, GEN b)
{
  pari_sp av = avma;
  GEN v;
  long n;
  n = itos(subii(b,a))+1;
  if(n<0) n=0;
  v = cgetg(n+1, t_VEC);
  return gc_GEN(av, v);
}
```

Finishing the code

Our test triggers a SEGV! The components of our vector are not initialised, so we created an invalid object.

Finish the code by filling the components:

```
GEN interval(GEN a, GEN b)
{
  pari_sp av = avma;
  GEN v;
  long n, i;
  n = itos(subii(b,a))+1;
  if(n<0) n=0;
  v = cgetg(n+1, t_VEC);
  gel(v,1) = a;
  for (i=2; i<=n; i++) gel(v,i) = addis(gel(v,i-1),1);
  return gc_GEN(av, v);
}
```

Thank you!

Have fun with Pari!